Discriminatively Trained Templates for 3D Object Detection: A Real Time Scalable Approach

Reyes Rios-Cabrera[a,b]
[a]CINVESTAV, Robotics and Advanced Manufacturing
Av. Industria Metalurgica 1062
Ramos Arizpe, 25900, Mexico
reyes.rios@cinvestav.edu.mx

Tinne Tuytelaars[b]
[b]KU Leuven, ESAT-PSI-VISICS, iMinds
Kasteelpark Arenberg 10
Leuven, B-3001, Belgium
tinne.tuytelaars@esat.kuleuven.be

Abstract

In this paper we propose a new method for detecting multiple specific 3D objects in real time. We start from the template-based approach based on the LINE2D/LINEMOD representation introduced recently by Hinterstoisser et al., yet extend it in two ways. First, we propose to learn the templates in a discriminative fashion. We show that this can be done online during the collection of the example images, in just a few milliseconds, and has a big impact on the accuracy of the detector. Second, we propose a scheme based on cascades that speeds up detection.

Since detection of an object is fast, new objects can be added with very low cost, making our approach scale well. In our experiments, we easily handle 10-30 3D objects at frame rates above 10fps using a single CPU core. We outperform the state-of-the-art both in terms of speed as well as in terms of accuracy, as validated on 3 different datasets. This holds both when using monocular color images (with LINE2D) and when using RGBD images (with LINEMOD). Moreover, we propose a challenging new dataset made of 12 objects, for future competing methods on monocular color images.

Figure 1. 1st row: our approach on heavily cluttered images (our dataset). 2nd row: it is running at 14fps detecting 30 objects simultaneously (dataset of [2]). 3rd row: our approach detecting objects' 3D pose (the rotation of each axis is shown in yellow, green and red for the middle object)(dataset of [7]).

1. Introduction

Recognition of specific objects, when compared to category-level recognition, may look like an easy task. However, methods are expected to be more efficient (detecting multiple objects in realtime) and more accurate (less false positives/false negatives, as well as more precise localization and pose estimation). Both are relevant e.g. in a robotic context. In general, methods in the literature can be split in local feature based methods, that work well on textured objects, and template-based methods, that work well on texture-poor objects. Here we follow the latter line of research, building on the work of Hinterstoisser *et al.* [5].

Template matching traditionally has been a popular method in manufacturing environments, where occlusions and clutter can mostly be avoided and lighting and pose variations can be carefully controlled, although it has also been applied in more challenging settings such as, e.g., pedestrian detection [4]. The main advantages of template matching are that it can be implemented efficiently and that it works well also for objects with few discriminating features that are dominantly determined based on their overall shape. While initially applied mostly in a relatively clean 2D setting, it has been shown that, by using multiple tem-

plates and robust representations, it can also be very successful for detecting 3D objects in more uncontrolled scenarios [5, 7]. Additionally, the use of templates also allows to transfer efficiently and accurately metadata that has been provided in an offline phase for the example images, to the object in the test image once it has been detected. This could be useful to indicate, e.g., possible grasping points or other anchor points to guide robot manipulation.

Current template based methods use greedy or heuristic approaches to learn the templates [2, 5, 6, 7, 13], which often produce suboptimal results. They cannot scale easily [5, 6, 7, 13] or are not fast enough [13], or cannot handle heavy clutter [2]. In this paper we propose to combine techniques that are usually used for offline learning methods, with techniques used in online learning ones, to overcome these limitations.

In particular, we build on the work of LINE2D/LINEMOD [5], yet extend it in several directions, applying ideas that have proven useful for category-level recognition to the setting of specific objects: adding a discriminative learning phase for better performance and adding a cascade for better efficiency. Discriminative learning is standard for category-level recognition, but not often used for specific objects – probably because i) it's often assumed all appearance changes can be modeled mathematically (e.g. modeling viewpoint changes with affine transformations), and ii) people usually target online schemes. Regarding the former argument, this only holds under certain restricting assumptions (e.g. planar or symmetric objects) or when a 3D model of the object is available - which usually is not the case. Starting from example images, this usually means that a large number of templates is needed, each of them covering only a small range of viewpoints or viewing conditions. Regarding the latter, the main idea is that one wants a fast system able to learn by simply showing it images of the object to be detected and discriminative learning is not a straight forward step because of lack of intraclass features. Here, we show that learning in a discriminative fashion what is really characteristic for the object at hand vs. what can easily be confused with other objects/background, has a strong impact on recognition performance. We also show that learning time can be kept low – in fact neglectable compared to the time needed to collect images of the object. We add three types of discriminative learning: 1) learning each template discriminatively, focusing on what is characteristic (we refer to this as DTT: Discriminatively Trained Templates), 2) using a boosting-scheme to learn weights for the different templates and how to combine their scores – in this step, we also create a cascade that speeds up the detection, and 3) tuning each template in terms of the number of non-zero bits. With these last two improvements added we refer to our method as DTT-OPT (DTT Optimized). Figure 1 shows some example detections obtained with our method.

This combination of online methods with learning based methods for specific object detection is our main contribution. Hinterstoisser *et al.* [5] and Damen *et al.* [2] explain the advantage of having an online learning system without the need of *costly* optimization. Another argument is that they do not need a training dataset, a step that requires extensive manual labeling. But why not optimize the templates obtained via the online learning ? This way we can get significant improvement in terms of accuracy and speed. Here we demonstrate how to accomplish that.

2. Related Work

The detection of instances of a specific object was not addressed for a while, since it was considered an easier problem (compared to category-level object detection), and traditional methods such as [1, 8, 11] seemed to work well as long as the objects have enough texture. However, in fields like robotics where a robot has to handle texture-poor objects and localize them in real-time, including their orientation, with high accuracy, and possibly their contours and segmentation, those traditional methods do not work well.

Texture-poor Object Detection: Addressing this issue, recently several new methods have been proposed for detection of 3D texture-poor objects. Hinterstoisser *et al.* [6] propose a method coined DOT (Dominant Orientation Templates). The method does not require a time consuming training stage, and can handle untextured objects. It is designed to be robust to small image transformations by spreading gradient orientations on a regular grid. Using binary operations, evaluation of the templates is very fast. Another advantage of the DOT detector, is that the Hamming distance between templates can be used to create clusters grouped by OR'ing similar templates. This allows for efficient branch and bound. However, the method becomes slow when handling several objects at the same time, and it is not template size invariant.

In [13], Steger proposed a method for object recognition aimed for industrial inspection. This method matches a 2D model of an object to an input test image and is able to recognize objects under similarity transformations at video rate (even back in 2002 when it was first proposed) by using a similarity measure that is inherently robust against occlusion, clutter, and nonlinear illumination changes. However, when using thousands of templates, as needed for 3D objects or when detecting multiple objects, its speed decreases, as demonstrated in [5], where it was compared to the LINE2D detector and was shown to be 100 times slower even when using already 4 pyramid levels for speeding up.

As DOT, the LINE2D method of [5] does not require a time consuming training stage and can handle untextured

objects. It is also based on dominant gradient orientations, it is robust to small image transformations and very robust to strong clutter. This is the type of templates that we mainly build on in this work (although the main idea can also be applied to other types of templates, e.g. DOT). Additionally, the authors of [5] show how to combine color images with a dense depth sensor, if available, by taking 3D surface normal orientations into account. That variation of the method is called LINEMOD, and we also perform experiments with it in this paper.

LINE2D shows a much higher accuracy than DOT. However it is also slower and, as DOT, it is not scalable for detection of many objects simultaneously. In [5], it was shown that LINE2D reduces greatly the false positive rate, having very few per image in spite of heavy clutter. However, the evaluation criterion used in this study was not very strict, as we discuss later in section 5.

Recently, a method based on LINEMOD [5] for the automatic modeling, detection, and tracking of 3D objects with RGBD (Red, Green, Blue plus Depth) sensors was proposed in [7] . They demonstrate how to build templates from 3D models, and how to estimate the 6 degrees-of-freedom pose accurately and in real-time. They use pose estimation and color information to check the detection hypotheses as a post-processing step. That improves the correct detection rate by 13% with respect to the original LINEMOD. The method achieves very accurate results. However, if we intend to handle the 15 objects it proposes simultaneously, the detection speed again falls to 0.72 fps using 2 cores. Another drawback is the need for a 3D model to train.

Recently, Damen *et al.* [2] presented a method for learning and detecting multiple texture-less 3D objects. The method they propose is able to run in real time, and was designed for video input. It is also scalable by implementing tractable extraction of edgelet constellations and using a library lookup based on rotation and scale-invariant descriptors. In their test setup, they evaluate multi-object detection on a 30 objects dataset showing detections in the order of milliseconds. They show how their method can scale keeping a high frame rate performance. However, their testing dataset has very small amounts of clutter. In their testing, they evaluate using the 50% intersection over union criterion, which we consider not precise enough for template matching methods. This approach has the advantage that it does not need to train all views or sizes as DOT/LINE2D/LINEMOD do. It can also handle many objects at the same time. It's the only method that can handle 5 objects or more simultaneously and still run at several frames per second. However the approaches based on oriented gradients are much more discriminant and can handle much higher amounts of background clutter. Moreover, even though the edgelets method runs in real time, it suffers an increase of testing time by 10 folds, when ambiguous objects are included.

Learning Based Methods: The literature of learning based methods is very wide and falls largely outside the scope of this paper. An example of such methods, trained to handle templates and metadata transfer, was proposed recently by Malisiewicz [9]. The method combines multiple SVMs, one per training sample. However, it is extremely slow and therefore not usable in a robotics context. Only recently [14] proposed an efficient template learning scheme, albeit in a different context (finegrained classification).

In [12], a method for learning an efficient multiview category-level object detector based on DOT [6] was proposed. To this end, the authors propose to first cluster the training examples using HOG, then generate a DOT template for each training example and learn a mask for it based on a linear SVM, so as to remove background noise while at the same time keeping the relevant context information. The different templates are then combined into a strong classifier using boosting. They also experiment with meta-data transfer. Here, we build on this work, using the same ideas of applying a linear SVM to discriminatively train the templates and using boosting to combine multiple templates into a cascade structure. However, we build on LINE2D instead of DOT. More importantly, we focus on specific object detection instead of category-level detection. While the high-level ideas are similar, the way they are employed for these different tasks are completely different.

Considering the analysis above, we can conclude that all methods have their own disadvantages. In the problem of specific object detection there is still room for improvement in terms of both speed, scalability and accuracy.

3. Background and notations

The LINE2D/LINEMOD method binarizes gradient orientations into a byte. It selects gradients based on their magnitudes. However, the location of each gradient is taken into account to avoid the problem of accumulating many gradient orientations in a local area. In order to binarize the gradient orientations, it defines a range from $[0 - 180^\circ]$ and it uses steps of $180^\circ/8 = 22.5^\circ$ to define the bit position it belongs to. For example an orientation of 44° would result into $[0000\ 0010]$. See [5] for a complete reference.

Then during testing only bitwise operations are used. This speeds up detection.

A model or template $\mathcal{T}$ can then be defined as a pair $\mathcal{T} = (\mathcal{O}, \mathcal{P})$, where $\mathcal{O}$ is a reference image of the object to detect, and $\mathcal{P}$ specifies a region in $\mathcal{O}$. The template can then be compared with a region at location c in a test image $\mathcal{I}$ based on the similarity measure proposed by Steger [13]:

$$\varepsilon_s(\mathcal{I}, \mathcal{T}, c) = \sum_{r \in \mathcal{P}} |\cos(\text{ori}(\mathcal{O}, r) - \text{ori}(\mathcal{I}, c + r))| \quad (1)$$

where $\text{ori}(\mathcal{O}, r)$ and $\text{ori}(\mathcal{I}, c+r)$ are the gradient orientations at location r in $\mathcal{O}$ and $c+r$ in $\mathcal{I}$ respectively. In [5], a variant of this measure is proposed, using the maximum over a small neighbourhood:

$$\varepsilon(\mathcal{I}, \mathcal{T}, c) = \sum_{r \in \mathcal{P}} \left(\max_{t \in \mathcal{R}(c+r)} |\cos(\text{ori}(\mathcal{O}, r) - \text{ori}(\mathcal{I}, t))| \right) \tag{2}$$

By finding the maximum, gradients in the template and test image get better aligned.

As in [12], we convert binary templates into weak classifiers. We define a weak classifier $\top_t(I)$ based on template $\mathcal{T}_t$ that classifies input image windows $I = (\mathcal{I}, c)$ as:

$$\top_t(I) = \begin{cases} +1 & if\ \varepsilon(\mathcal{I}, \mathcal{T}_t, c) \geq \tau_t \\ -1 & otherwise \end{cases} \tag{3}$$

Based on the binary response of $\top_t(I)$ for a pool of templates $\mathcal{T}_t$, we then build a strong classifier $H(I)$:

$$H(I) = sign\left(\sum_{t=1}^{T_0} \alpha_t \top_t(I) \right) \tag{4}$$

$H(I)$ builds on T_0 templates selected by AdaBoost. Weights α_t and thresholds τ_t (used in $\top_t(I)$) are set automatically for each template using the standard AdaBoost procedure.

4. Description of Our Method

In our proposal we try to reconcile the advantages of traditional offline learning-based methods that use big annotated datasets with the advantages of online learning methods. To this end, we follow 3 main steps: **i)** we propose to use a small validation set constructed from negative images. Since acquiring a set of negative images is not costly at all, we use one to learn better templates with a linear SVM, building on the method of [12]. **ii)** Then we create a cascaded version to further speed up the detection process. **iii)** Finally, we further tune each template using the negative samples, i.e. we decide which bits should be kept, and how many of them, starting from the weights learnt by the linear SVM. Contrary to the template based methods of [5, 6, 7], we tune each of the templates separately because using the same parameters (number of regions) for every single template is suboptimal. Each of these steps is described in the sections below. We show how to optimize them quickly, keeping online learning speeds.

4.1. DTT: Discriminatively Trained Templates

Inspired by [12], we propose to learn the most important elements of a template by using the weights of a linear SVM. We emphasize the fact that once we have learned the weights, we binarize the values so we can directly use it as a template. This is done before learning a cascade structure to further speed up (see section 4.2). Hence using the SVM for training does not take extra time during testing, as shown in [12]. In [10] it was shown experimentally that feature selection using weights from linear SVMs yields better classification performance than other feature weighting methods. A Support Vector Machine trains a linear classifier of the form $sgn(w^T x + b)$. Learning is addressed as an optimization problem with the goal of maximizing the margin, i.e., the distance between the separating hyperplane $w^T x + b = 0$ and the nearest training vectors. We use of the weights of a linear classifier to discriminate which regions of the templates are most important to compare with a test sample, and which are actually damaging the performance.

The elements of the weights vector w that are negative are considered to be damaging the template performance, since those were generated mainly by support vectors from the negatives. We use a threshold on the elements of w so as to select only those elements that contribute most positively to the object detection.

Training the Templates We use a precomputed set of 10,000 negative samples, from 100 cluttered images that do not contain the objects we want to train. To apply the scheme above and select the most relevant elements of the templates as proposed in [12], we need a set of positive samples as well. How these are extracted is explained below.

Considering online learning, for each template t_a of an image a (containing homogeneous background) we follow the next steps:

1. We capture the input image to learn the template t_a. We set a regular grid for the training image and compute the gradients. As in LINE2D, to each location of the image we assign the gradient whose quantized orientation occurs most often in a 3×3 neighborhood. Here we emphasize that we do not use the regular grid when testing but only for selecting the most discriminative gradient orientations in the training process.

2. Then we obtain all the strongest unique gradient orientations located in the cells (up to b_n) that are above a threshold. This is done homogeneously on the input image (as opposed to the approach in [5, 12], where only the strongest top n_0 gradient orientations are chosen). We do not select the top n_0 gradient orientations, because it is possible that all the top n_0 could be associated with the same gradient orientation. We observe that this can produce redundancy of orientations in the LINE2D method.

3. Then we find up to P nearest neighbors (NN) among the previously trained templates, that will be used as

additional positive examples for learning the linear SVM (if no templates were learnt yet, we use only the current one). For this, we compare templates using the similarity measurement of equation 1. We keep only those nearest neighbors that have a similarity above 90%. For each NN, we obtain the orientations as in step 2.

4. We select a random set of a few hundred negatives from the large pool of negatives, (precomputed with the gradient spreading algorithm of [5]).

5. Then we AND the template gradients of t_a with each of the NN, to produce positive training vectors for the SVM, and with the selected negative subset, to produce negative training vectors.

6. A linear SVM is trained, and the resulting weights w are used to select the R most discriminative regions.

4.2. DTT-OPT: Optimized Scheme

For each object separately, once we have trained the templates, we create C clusters. We use the bottom-up clustering method suggested in [6], but computing similarity with eq 1. This clustering takes only few milliseconds. For each cluster, we construct a strong classifier using AdaBoost. This focuses only on the templates of group C_i. Figure 2 shows this procedure. Then we convert each strong classifier into a cascade by using Multiple Instance Pruning (MIP) proposed in [15]. This procedure takes about 2 seconds per cluster. We use in practice 12 clusters per object. MIP ensures that all positive training samples are correctly classified. Once we have created the cascades, we can detect the objects (right part of Figure 2). Only if the object makes it to the end of the cascade, we proceed to compare with the whole tuned look up table of cluster C_i as in the traditional template based methods.

For the final clustered table, we tune each template t_a to keep $R \pm r$ bits. We calculate the similarity using equation 1 for each of the 10,000 negative samples. If the similarity is bigger than a threshold value (65%), we count it as an error. We decrease the number of bits (eliminating first the ones with smaller weights) until we reach a target error, or a minimum allowed number of regions $R - r$. We refer to the optimized version as DTT-OPT.

5. Experimental Results

We evaluate our method on 3 datasets: i) our data set [1] consisting of 12 objects, ii) a dataset consisting of 30 objects to be handled simultaneously [2], and iii) a 15 objects data set, consisting of 3D models [7], where not only detection of the object is evaluated, but also its 3D pose.

[1] Available online, contact the authors for a link

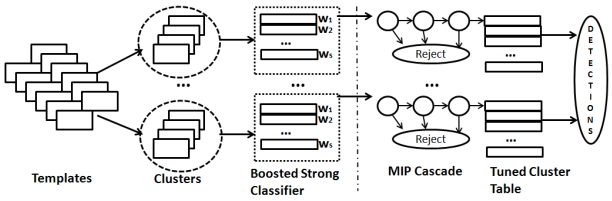

Figure 2. We cluster the templates into C groups. For each we learn a strong classifier and create a MIP cascade. Only hypothesis passing the cascade are compared with the tuned cluster table.

Figure 3. The 12 objects of our dataset.

5.1. Our 12-Object Dataset

We created a dataset of about 1K images per object for testing (12,530 images for the 12 objects). The testing images include heavy clutter, changes in illumination, and occlusion. Figure 1 first row, shows 2 test images. In order to create the dataset, we first determined an approximate location of the objects by using a calibration pattern. Then we ran LINE2D very densely only in the approximate location to determine a tight bounding box. Once we had this preliminary annotations, we inspected the images manually and corrected the errors.

We trained about 1000 templates for each object, covering sizes [1.0:2.0], 360 degrees rotation, 30-85 degrees of tilt rotation and $\pm$ 45 degrees of in-plane rotation. During training it takes on average about 60msec to optimize a template. We can train at 10fps while showing the object to the camera, and learning only a new template when this is 6% different from all saved ones. Once the object is learnt, we need 24s for creating the cascades using a single core.

For evaluation, contrary to [5], we compute the precision-recall curves for each experiment. We use this type of evaluation, because it allows a more accurate comparison than selecting only the top 1 or top N detections of each image.

Moreover, if we only get the top N detections from each image, then comparing detections with other images is inaccurate. To be accurate, we need to retrieve as many detections as there are available and sort the scores over images (some test images might be much harder than others). Then we can accurately compute the performance of detections.

To compute the Area under the curve (AuC) we calculate

the area of the polygon below each curve. In the original LINE2D testing dataset [5], the authors consider the object to be correctly detected if the predicted location was within a fixed radius of the ground truth annotations (specified as a single coordinate). That approximation has a serious drawback. The location is only approximated to the centroid of the object. However the size of the bounding box and the aspect ratio can vary drastically, and this is not evaluated with this measure.

Since template based detectors try to find the best template fit, here we set a minimum bounding box overlap of 70% intersection over union. Inspecting the bounding boxes, and the associated template for each detection, we found out that 70% was a fair choice to represent the object. We show results of our methods DTT and DTT-OPT, compared to LINE2D, in Figure 4. Detecting the 12 objects simultaneously, LINE2D runs at 1.4fps using 2 pyramid levels and 2 CPU cores. DTT-OPT runs at 11.25fps using a single core to detect the objects simultaneously, see Table 1.

Method	AuC	At 70% recall	Speed	CPU
DTT-OPT	85.5%	83.9% precision	11.25fps	i7@2.8GHz, 8GB RAM (1core)
LINE2D	73.6%	68.8% precision	1.4fps	i7@2.8GHz, 8GB RAM (2 cores)

Table 1. Comparing our method with LINE2D [5] in our proposed dataset. The AuC is averaged over the 12 objects. The speed is measured at 70% recall. Our method improves 11.9% in average AuC, and 15.1% in precision at 70% recall, while being 8 times faster using half the number of cores.

5.2. Dataset of 30 Textureless Objects

We also evaluate on the data set proposed by Damen *et al.* [2]. Since our method needs explicit templates for different views and sizes, we rotate each training image from the dataset and increase their sizes in steps of 12% to train in total about 32,000 templates for the whole set of 30 objects. We learn a new template, only when it was at least 10% different from the already learnt ones. See Table 2.

Method	At 50% recall	Speed	CPU
DTT-OPT	90% precision	14.3fps	i7@2.8GHz, 8GB RAM (1core)
LINE2D	80% precision	4.1fps	i7@2.8GHz, 8GB RAM (2 cores)
Damen *et al*	75% precision	7fps	2.53Hz, 6GB RAM (1 core)

Table 2. Comparing our method with LINE2D [5] and Damen *et al.* [2]. Our method outperforms the state of the art methods in both speed and accuracy.

In our testing we search for all 30 objects, apply non-maximum suppression and select the winning hypothesis by selecting the biggest object found (if it is greater than a threshold). This is because some templates appear to be subparts of bigger objects. At a 50% recall, LINE2D runs on average at 4.1fps. This is using 2 pyramid levels and 2 cores. This is faster than when we tested on heavily cluttered images using more than double templates. The reason is because the testing images have very few/no clutter.

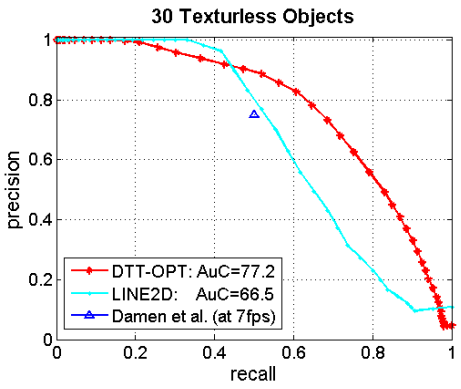

Figure 5. Results on 30 textureless objects detected simultaneously. Damen *et al.* only present results at 50% recall.

Damen *et al.* use a single core and their system runs at 7fps at 50% recall. Our DTT-OPT runs at 14.3fps (average) using a single core, with a higher precision (90%). Figure 5 shows the precision recall curves for the three methods.

Approach	DTT-3D	Hinterstoisser[7]	Drost [3]
Sequence	Matching Score/Speed		
Ape	95.0% / 55.8ms	95.8% / 127ms	86.5% / 22.7s
Bench Vise	98.9% / 53.3ms	98.7% / 115ms	70.7% / 2.94s
Driller	94.3% / 54.6ms	93.6% / 121ms	87.3% / 2.65s
Cam	98.2% / 58.4ms	97.5% / 148ms	78.6% / 2.81s
Can	96.3% / 55.3ms	95.4% / 122ms	80.2% / 1.60s
Iron	98.4% / 54.3ms	97.5% / 116ms	84.9% / 3.18s
Lamp	97.9% / 54.8ms	97.7% / 125ms	93.3% / 2.29s
Phone	95.3% / 58.4ms	93.3% / 157ms	80.7% / 4.70s
Cat	99.1% / 53.5ms	99.3% / 111ms	85.4% / 7.52s
Hole Puncher	97.5% / 54.2ms	95.9% / 110ms	77.4% / 8.30s
Duck	94.2% / 53.6ms	95.9% / 104ms	40.0% / 6.97s
Cup	97.5% / 54.1ms	97.1% / 105ms	68.4% / 16.7s
Bowl	99.7% / 51.50ms	99.9% / 97ms	95.7% / 5.18s
Box	99.8% / 56.0ms	99.8% / 101ms	97.0% / 2.94s
Glue	96.3% / 58.5ms	91.8% / 135ms	57.2% / 4.03s
Average	**97.2% / 55.1ms**	96.6% / 119ms	79.3% / 6.3s

Table 3. We use the same evaluation criteria as in [7]. Computing the gradients and the normals takes 30ms and 12 ms. Our reported times are for the whole pipeline using one core. [7] uses 2 pyramid levels and needs one core for each level. Using their system to detect simultaneously all 15 objects, we can subtract the time for processing the gradients/normals which is computed only once. Their system can then run at 0.72fps using 2 cores, while ours runs at 4.2fps using a single core, which makes us roughly 10 times faster.

5.3. 3D Model Based Dataset of 15 Objects

Finally, we use the RGB-D dataset proposed in [7], where Hinterstoisser *et al.* improved LINEMOD by adding 2 post-processing steps that verify each hypothesis further. One of them is a color checking step where the hypotheses are compared with the original object color (difference of its hue). The other is a depth points checking. We use the same improvements as post-processing step. We also use

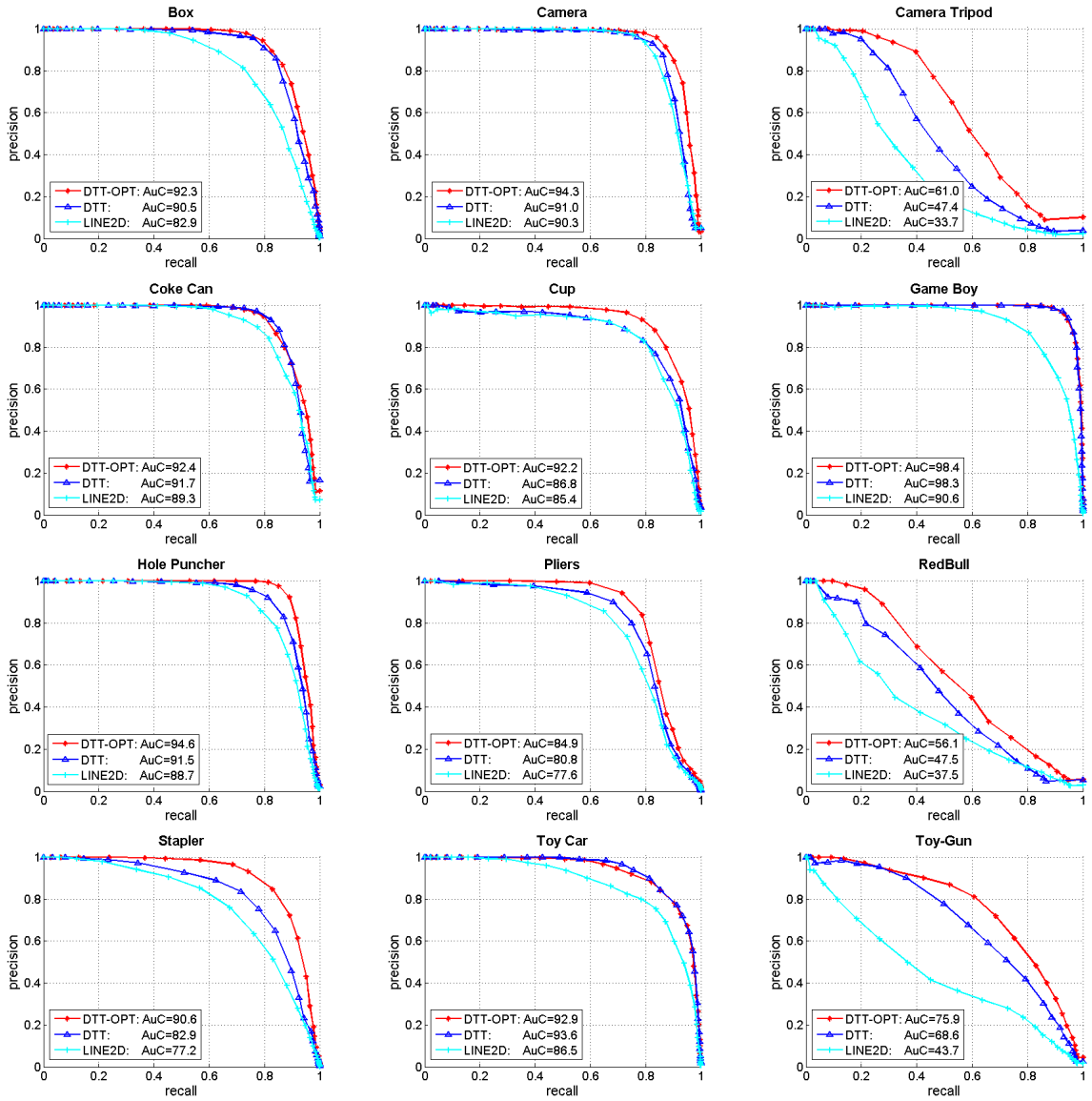

Figure 4. We tested our method on 12 objects comparing with the LINE2D baseline. We tested 2 improvements. One where we trained the template discriminatively called DTT, and a second one, where each template is tuned, and then a a cascaded strong classifier for different clusters is added. This is called DTT-OPT.

the recommended parameters (rotation sampling of 15 degrees and scale sampling of 10 cm). However to train our objects, we select the gradients and normals, using our discriminatively trained method. During detection, we use the clustered cascades to speed up detection, and the tuned tables, as described before. We call this method: DTT-3D. Figure 6 right column shows some detection examples of our method. During training, we need on average 62msec for each template, plus about 24 seconds to create the speed up cascades. In total, this brings us up to 217 seconds for training an object. Compared to [7] where training is up to 54 seconds (for "bech vise"), it is still relatively fast, especially if one considers that this is done only once per object. Results can be seen in table 3. While testing, our method is almost an order of magnitude faster when handling 15 objects simultaneously, and more accurate.

6. Conclusions

In the field of specific object detection, there is still plenty of room for improvement. One of the standing problems is the need of several templates for training. Another is the need to model different sizes. Nevertheless, our proposed DTT-OPT and DTT-OPT-3D methods improve the current state of the art in both speed and accuracy when

Figure 6. Left: Detection examples of 12 objects detected simultaneously. Middle: Detections of 30 textureless objects (simultaneously). Right: Detections of the 3D dataset. We show the object axes in z-yellow, x-red, y-green.

using monocular color images as well as RGBD images. Training discriminatively each template allows an increment in detection performance. By combining techniques usually used in offline learning methods with online learning ones, we can obtain a faster and more accurate detector.

References

[1] H. Bay, A. Ess, T. Tuytelaars, and L. J. V. Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346–359, 2008. 2

[2] D. Damen, P. Bunnun, A. Calway, and W. Mayol-Cuevas. Real-time learning and detection of 3d texture-less objects: A scalable approach. In *British Machine Vision Conference (BMVC)*. BMVA, September 2012. 1, 2, 3, 5, 6

[3] B. Drost, M. Ulrich, N. Navab, and S. Ilic. Model globally, match locally: Efficient and robust 3d object recognition. In *Computer Vision and Pattern Recognition, CVPR 2010*, pages 998–1005, 2010. 6

[4] D. Gavrila and V. Philomin. Real-time object detection for smart vehicles. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 87–93, 1999. 1

[5] S. Hinterstoisser, C. Cagniart, S. Ilic, P. Sturm, N. Navab, P. Fua, and V. Lepetit. Gradient response maps for real-time detection of texture-less objects. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 2012. 1, 2, 3, 4, 5, 6

[6] S. Hinterstoisser, V. Lepetit, S. Ilic, P. Fua, and N. Navab. Dominant orientation templates for real-time detection of texture-less objects. In *IEEE Computer Vision and Pattern Recognition (CVPR)*, pages 2257–2264, 2010. 2, 3, 4, 5

[7] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, , and N. Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian Conference on Computer Vision (ACCV)*, 2012. 1, 2, 3, 4, 5, 6, 7

[8] D. G. Lowe. Object recognition from local scale-invariant features. In *IEEE International Conference on Computer Vision*, ICCV, pages 1150–, 1999. 2

[9] T. Malisiewicz, A. Gupta, and A. A. Efros. Ensemble of exemplar-svms for object detection and beyond. In *IEEE Int. Conf. on Computer Vision (ICCV)*, 2011. 3

[10] D. Mladenić, J. Brank, M. Grobelnik, and N. Milic-Frayling. Feature selection using linear classifier weights: interaction with classification models. In *Conf. on Research and development in information retrieval*. ACM, 2004. 4

[11] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *IEEE Conference on Computer Vision and Pattern Recognition - CVPR*, 2006. 2

[12] R. Rios-Cabrera and T. Tuytelaars. Boosting binary masks and dominant orientation templates for efficient object detection. *Under revision CVIU*. 3, 4

[13] C. Steger. Occlusion, clutter, and illumination invariant object recognition. In *International Archives of Photogrammetry and Remote Sensing*, volume XXXIV, part 3A, pages 345–350, 2002. 2, 3

[14] S. Yang, L. Bo, J. Wang, and L. G. Shapiro. Unsupervised template learning for fine-grained object recognition. In *NIPS*, pages 3131–3139. 3

[15] C. Zhang and P. A. Viola. Multiple-instance pruning for learning efficient cascade detectors. In *NIPS*, 2007. 5